Refining the design of a contracting finite-state dependency parser

Yli-Jyrä, Anssi Mikael

ACL Anthology
2012-07-23

Yli-Jyrä , A M , Piitulainen , J & Voutilainen , A 2012 , Refining the design of a contracting finite-state dependency parser . in Proceedings of FSMNLP 2012 . vol. 2012 , ACL Anthology , The International Workshop on Finite State Methods and Natural Language Processing 2012 - FSMNLP 2012 , Donostia - San Sebastian , Spain , 23/07/2012 . < http://ixa2.si.ehu.es/fsmnlp2012/ >

http://hdl.handle.net/10138/39236

submittedVersion

Refining the Design of a Contracting Finite-State Dependency Parser

Anssi Yli-Jyrä and **Jussi Piitulainen** and **Atro Voutilainen**
The Department of Modern Languages
PO Box 3
00014 University of Helsinki
{anssi.yli-jyra,jussi.piitulainen,atro.voutilainen}@helsinki.fi

Abstract

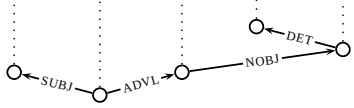
This work complements a parallel paper of a new finite-state dependency parser architecture (Yli-Jyrä, 2012) by a proposal for a linguistically elaborated morphology-syntax interface and its finite-state implementation. The proposed interface *extends* Gaifman's (1965) classical dependency rule formalism by separating lexical word forms and morphological categories from syntactic categories. The separation lets the linguist take advantage of the morphological features in order to reduce the number of dependency rules and to make them lexically selective. In addition, the relative functional specificity of parse trees gives rise to *a measure of parse quality*. By filtering worse parses out from the parse forest using finite-state techniques, the best parses are saved. Finally, we present *a synthesis of strict grammar parsing and robust text parsing* by connecting fragmental parses into trees with additional linear successor links.

1 Introduction

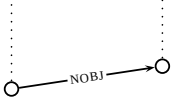
Finite-state dependency parsing aims to combine dependency syntax and finite-state automata into a single elegant system. Deterministic systems such as (Elworthy, 2000) are fast but susceptible to garden-path type errors although some ambiguity is encoded in the output. Some other systems such as (Oflazer, 2003; Yli-Jyrä, 2005) carry out full projective dependency parsing while being much slower, especially if the syntactic ambiguity is high. In the worst case, the size of the minimal finite-state automaton storing the forest is exponentially larger than the sentence: an 80-word sentence has potentially 1.1×10^{62} unrooted unlabeled dependency trees that are stored "compactly" into a finite-state lattice that requires at least 2.4×10^{24} states, see Table 4 in Yli-Jyrä (2012).

A truly compact representation of the parse forest is provided by an interesting new extended finite-state parsing architecture (Yli-Jyrä, 2012) that first recognizes the grammatical sentences in quadratic time and space if the nested dependencies are limited by a constant (in cubic time if the length of the sentence limits the nesting). The new system (Yli-Jyrä, 2012) replaces the additive (Oflazer, 2003) and the intersecting (Yli-Jyrä, 2005) validation of dependency links with reductive validation that gradually contracts the dependencies until the whole tree has been reduced into a trivial one. The idea of the contractions is illustrated in Example 1. In practice, our parser operates on bracketed trees (i.e., strings), but the effect will be similar.

(1) a. *time flies like an arrow*



b. *time flies like* *an arrow*



c. *time flies like an arrow*

Despite being non-deterministic and efficient, there are two important requirements that are not fulfilled by the core of the new architecture (Yli-Jyrä, 2012):

1. A mature finite-state dependency parser must be robust. The outputs should not be restricted to complete grammatical parses. For example, Oflazer (2003) builds fragmental parses but later drops those fragmental parses for which there are alternative parses with fewer fragments. However, his approach handles only gap-free bottom-up fragments and optimizes the number of fragments by a counting method whose capacity is limited.

2. Besides robustness, a wide-coverage parser should be able to assign reasonably well-motivated syntactic categories to every word in the input. This amounts to having a morphological guesser and an adequate morphology-syntax interface. Most prior work trivializes the complexity of the interface, being comparable to Gaifman's (1965) legacy formalism that is mathematically elegant but based on word-form lists. A good interface formalism is provided, e.g., by Constraint Grammar parsers (Karlsson et al., 1995) where syntactic rules can refer to morphological features. Oflazer (2003) tests morphological features in complicated regular expressions. The state complexity of the combination of such expressions is, however, a potential problem if many more rules would be added to the system.

This paper makes two main contributions:

1. It adapts Gaifman's elegant formalism to the requirements of morphologically rich languages. With the adapted formalism, grammar writing becomes easier. However, efficient implementation of the rule lookup becomes inherently trickier because testing several morphological conditions in parallel increases the size of the finite-state automata. Fortunately, the new formalism comes with an efficient implementation that keeps the finite-state representation of the rule set as elegant as possible.

2. The paper introduces a linguistically motivated ranking for complete trees. According to it, a tree is better than another tree if a larger proportion of its dependency links is motivated by the linguistic rules. In contrast to Oflazer (2003), our method counts the number of links needed to connect the fragments into a spanning tree. Moreover, since such additional links are indeed included in the parses, the ranking method turns a grammar parser into a robust text parser.

The paper is structured as follows. The next section will give an overview of the new parser architecture. After it, we present the new morphology-syntax interface in Section 3 and the parse ranking method in Section 4. The paper ends with theoretical evaluation and discussion about the proposed formalism in Section 5.

2 The General Design

2.1 The Internal Linguistic Representation

We need to define a string-based representation for the structures that are processed by the parser. For this purpose, we encode the dependency trees and then augment the representation with morphological features.

Dependency brackets encode dependency links *between* pairs of tokens that are separated by an (implicit) token boundary. The four-token string `abcd` has 12 distinct *undirected unlabeled dependency bracketings* `a(((b)c)d`, `a((b())c)d`, `a(()b()c)d`, `a(()bc())d`, `a(()b)c()d`, `a(b(()c))d`, `a(b(c()))d`, `a(b()c())d`, `a(b())c()d`, `a()b(()c)d`, `a()b(c())d`, `a()b()c()d`.[1]

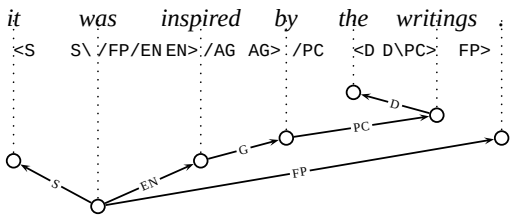
The basic dependency brackets extend with labels such as in `(LBL LBL)` and directions such as in `<LBL LBL\` and in `/LBL LBL>`. Directed dependency links designate one of the linked words as the head and another as the dependent. The extended brackets let us encode a full dependency tree in a string format as indicated in (2).[2] The dependent word of each

[1] Dependency bracketing differs clearly from *binary phrase-structure bracketings* that put brackets *around* phrases: the string `abcd` has only five distinct bracketings `((ab)(cd))`, `(((ab)c)d)`, `((a(bc))d)`, `(a((bc)d))`, and `(a(b(cd)))`.

[2] The syntactic labels used in this paper are: `AG`=Agent, `by`=Preposition 'by' as a phrasal verb complement, `D`=Determiner, `EN`=Past Participle, `FP`=Final Punctuation, `P`=adjunctive preposition, `PC`=Preposition Complement, `S`=Subject, `sgS`=Singular Subject.

link is indicated in trees with an arrowhead but in bracketing with an angle bracket.

(2)



In Table 1, the dependency bracketing is combined with a common textual format for morphological analyses. In this format, the base forms are defined over the alphabet of orthographical symbols Ω whereas the morphological symbols and syntactic categories are multi-character symbols that belong, respectively, to the alphabets Π and Γ. In addition, there is a token boundary symbol #.

Table 1: One morpho-syntactic analysis of a sentence

```
1 i t           PRON NOM SG3   <S          #
2 b e           V PAST SG13    S\ /FP /EN  #
3 i n s p i r e EN             EN> /AG     #
4 b y           PREP           AG> /PC     #
5 t h e         DET SG/PL      <D          #
6 w r i t i n g N NOM PL       D\ PC>      #
7 .             PUNCT          FP>         #
```

Depending on the type of the languages, one orthographical word can be split into several parts such as the inflectional groups in Turkish (Oflazer, 2003). In this case, a separate word-initial token boundary can be used to separate such parts into lines of their own.

The current dependency bracketing captures projective and weakly non-projective (1-planar) trees only, but an extended encoding for 2-planar and multi-planar dependency trees seems feasible (Yli-Jyrä, 2012):

2.2 The Valid Trees

We are now going to define precisely the semantics of the syntactic grammar component using finite-state relations.

The *finite-state languages* will be defined over a finite alphabet Σ and they include all finite subsets of the universal language Σ^*. The *(binary) finite-state relations* are defined over Σ^* and include all finite subsets of $\Sigma^* \times \Sigma^*$. In addition, they are closed under the operations over finite-state languages L and M and finite-state relations R and S according to Table 2. The language relation $\mathrm{Id}(L)$ restricts the identity relation to a language L. The composition of language relations corresponds to the intersection of their languages.

Table 2: The relevant closure properties

language	relation	meaning
LM	RS	concatenation
L^*	R^*	(Kleene) star
L^+	R^+	(Kleene) plus
$L \cup M$	$R \cup S$	union
	$\mathrm{Id}(L)$	language relation
$\mathrm{Id}^{-1}(R)$		L for $R = \mathrm{Id}(L)$
$L - M$	$\mathrm{Id}(L) - \mathrm{Id}(M)$	set difference
	$L \times M$	cross product
	$R_{\mid L}$	input restriction
	$R \circ S$	composition
	R^{-1}	inverse
	$\mathrm{Proj}_1(R)$	Id(the input side of R)
	$\mathrm{Proj}_2(R)$	Id(the output side of R)

For notational convenience, the empty string is denoted by ϵ. A string x is identified with the singleton set $\{x\}$.

The syntactic component of the grammar defines a set of parse strings where the bracketing is a valid dependency tree. In these parses, there is no morphological information. One way to express the set is to intersect a set of constraints as in (Yli-Jyrä, 2005). However, the contracting dependency parser expresses the Id relation of the set through a composition of simple finite-state relations:

$$\mathrm{Syn}_t = \mathrm{Proj}_1(\mathrm{Abst} \circ \underbrace{R \circ \ldots \circ R}_{t} \circ \mathrm{Root}), \quad (1)$$

$$\mathrm{Root} = \mathrm{Id}(\#). \quad (2)$$

In (1), Abst is a relation that removes all non-syntactic information from the strings,

$$\mathrm{Abst} = (\mathrm{Id}(\Gamma) \cup \mathrm{Id}(\#) \cup \mathrm{Delete})^*, \quad (3)$$

$$\mathrm{Delete} = \{(x, \epsilon) \mid x \in \Omega \cup \Pi\}, \quad (4)$$

and R is a relation that performs one layer of contractions in dependency bracketing.

$$\mathrm{R} = (\mathrm{Id}(\Gamma) \cup \mathrm{Id}(\#) \cup \mathrm{Left} \cup \mathrm{Right})^*, \quad (5)$$

$$\mathrm{Left} = \{(\texttt{<}\alpha\ \#\ \alpha\texttt{\textbackslash}, \epsilon) \mid \texttt{<}\alpha, \alpha\texttt{\textbackslash} \in \Gamma\}, \quad (6)$$

$$\mathrm{Right} = \{(\texttt{/}\alpha\ \#\ \alpha\texttt{>}, \epsilon) \mid \texttt{/}\alpha, \alpha\texttt{>} \in \Gamma\}. \quad (7)$$

The parameter t determines the maximum number of layers of dependency links in the validated bracketings. The limit of Syn_t as t approaches ∞ is not necessarily a finite-state language, but it remains context-free because only projective trees are assigned to the sentences.

2.3 The Big Picture

We are now ready to embed the contraction based grammar into the bigger picture.

Let $x \in \Omega^*$ be an orthographical string to be parsed. Assume that it is segmented into n tokens. The string x is parsed by composition of four relations: the relation $\{(x, x)\}$, the lexical transducer (Morph), the morphology-syntax interface (Iface), and the syntactic validator Syn_{n-1}.

$$\mathrm{Parses}(x) = \mathrm{Id}(x) \circ \mathrm{Morph} \circ \mathrm{Iface} \circ \mathrm{Syn}_{n-1}. \quad (8)$$

The language relation $\mathrm{Proj}_2(\mathrm{Parses}(x))$ encodes the parse forest of the input x.

In practice, the syntactic validator Syn_{n-1} cannot be compiled into a finite-state transducer due to its large state complexity. However, when each copy of the contracting transducer R in (1) is restricted by its admissible input-side language, a compact representation for the input-side restriction $(\mathrm{Syn}_{n-1})_{|X}$ where $X = \mathrm{Proj}_2(\mathrm{Id}(x) \circ \mathrm{Morph} \circ \mathrm{Iface})$ is computed efficiently as described in (Yli-Jyrä, 2012).

3 The Grammar Formalism

In the parser, the linguistic knowledge is organized into Morph (the morphology) and Iface (the lexicalized morphology-syntax interface), while Syn has mainly a technical role as a tree validator. Implementing the morphology-syntax interface is far from an easy task since it is actually the place that lexicalizes the whole syntax.

3.1 Gaifman's Dependency Rules

Gaifman's legacy notation (Gaifman, 1965; Hays, 1964) for dependency grammars assigns word forms to a finite number of potential *morpho-syntactic* categories that relate word forms to their syntactic functions. The words of particular categories are then related by *dependency rules*:

$$X_0(X_p, \ldots, X_{-1}, \texttt{*}, X_1, \ldots, X_m). \quad (9)$$

The rule (9) states that a word in category X_0 is the head of dependent words in categories $X_p, \ldots, X_{-1}$ before it and words in categories $X_1, \ldots, X_m$ after it, in the given order. The rule expresses, in a certain sense, the frame or the argument structure of the word. Rule $X(\texttt{*})$ indicates that the word in category X can occur without dependents.

In addition, there is a *root rule* $\texttt{*}(X)$ that states that a word in category X can occur independently, that is, as the root of the sentence.

In the legacy notation, the distinction between complements and adjuncts is not made explicit, as both need to be listed as dependents. To compact the notation, we introduce *optional dependents* that will be indicated by categories $X_p\texttt{?}, \ldots, X_{-1}\texttt{?}$ and categories $X_1\texttt{?}, \ldots, X_m\texttt{?}$. This extension potentially saves a large number of rules in cases where several dependents are actually adjuncts, some kinds of modifiers.[3]

3.2 The Decomposed Categories

In practice, atomic morpho-syntactic categories are often too coarse for morphological description but too refined for convenient description of syntactic frames. A practical description requires a more expressive and flexible formalism.

In our new rule formalism, each morpho-syntactic category X is viewed as a combination of a morphological category M (including the information on the lexical form of the word) and a syntactic category S. The morphological category M is a string of orthographical and morphological feature labels while S is an atomic category label.

The morphological category M_0 and the syntactic category S_0 are specified for the head of each dependency rule. Together, they specify the morpho-syntactic category (M_0, S_0). In contrast, the rule specifies only the syntactic categories $S_p, \ldots, S_{-1}$,

[3] Optional dependents may be a worthwhile extension even in descriptions that treat the modified word as a complement of a modifier.

and $S_1, \ldots, S_m$ of the dependent words and thus delegates the selection of the morphological categories to the respective rules of the dependent words. The categories $S_p, \ldots, S_{-1}$, and $S_1, \ldots, S_m$ may again be marked optional with the question mark.

The rules are separated according to the direction of the head dependency. Rules (10), (11) and (12) attach the head to the right, to the left, and in any direction, respectively. In addition, the syntactic category of the root is specified with a rule of the form (13).

$$\rightarrow S_0(S_p, \ldots, S_{-1}, *[M_0], S_1, \ldots, S_m), \quad (10)$$

$$\leftarrow S_0(S_p, \ldots, S_{-1}, *[M_0], S_1, \ldots, S_m), \quad (11)$$

$$S_0(S_p, \ldots, S_{-1}, *[M_0], S_1, \ldots, S_m), \quad (12)$$

$$*(S_0). \quad (13)$$

The interpretations of rules (10) - (12) are similar to rule (9), but the rules are lexicalized and directed. The feature string $M_0 \in (\Omega^*\%\Omega^* \cup \Omega^*)\Pi^*$ defines the relevant head word forms using the features provided by Morph. The percent symbol (%) stands for the unspecified part of the lexical base form.

The use of the extended rule formalism is illustrated in Table 3. According to the rules in the table, a phrase headed by preposition *by* has three uses: an adjunctive preposition (P), the complement of a phrasal verb (by), or the agent of a passive verb (AG). Note that the last two uses correspond to a fully lexicalized rule where the morphological category specifies the lexeme. The fourth rule illustrates how morphological features are combined in N NOM SG and then partly propagated to the atomic name of the syntactic category.

Table 3: Extended Gaifman rules

```
1 P (*[% PREP], PC)                 % prepos.
2 by (*[by PREP], PC)               % phrasal
3 AG (*[by PREP], PC)               % agent
4 sgS (D?, M?, *[% N NOM SG], M?) % noun
```

3.3 Making a Gaifman Grammar Robust

Dependency syntax describes complete trees where each node is described by one of the dependency rules. Sometimes, however, no complete tree for an input is induced by the linguistically motivated dependency rules. In these cases, only tree fragments can be motivated by the linguistic knowledge. To glue the fragments together, we interpret the roots of fragments as *linear successors* – thus dependents – for the word that immediately precedes the fragment.

The link to a linear successor is indicated with a special category ++ having a default rule ++(*). Since any word can act as a root of a fragment, every word is provided with this potential category. In addition, there is, for every rule (12), an automatic rule ++$(S_p, \ldots, S_{-1}, *[M], S_1, \ldots, S_m)$ that allows the roots of the fragments to have the corresponding dependents. Similar automatic rules are defined for the directed rules.

The category ++ is used to indicate dependent words that do not have any linguistically motivated syntactic function. The root rule *(++) states that this special category can act as the root of the whole dependency tree. In addition to the root function expressed by that rule, an optional dependent ++? is appended to the end of every dependency rule. This connects fragments to their left contexts.

With the above extensions, all sentences will have at least one complete tree as a parse. A parse with some dependents of the type ++ are linguistically inferior to parses that do not have such dependents or have fewer of them. Removing such inferior analyses from the output of the parser is proposed in Section 4.

3.4 The Formal Semantics of the Interface

Let there be r dependency rules. For each rule i, $i \in \{1, ..., r\}$ of type (10), let

$$F_i = M_0, \quad (14)$$

$$G_i = S_{-1}\backslash \ \ldots S_p\backslash \ S_0> \ /S_m ... /S_1, \quad (15)$$

where $S_{-1}\backslash, \ldots, S_p\backslash, S_0>, /S_m, \ldots, /S_1 \in \Gamma$. For each rule of type (11), $S_0>$ in (15) is replaced with $<S_0$. Rules with optional dependents are expanded into subrules, and every undirected rule (12) splits into two directed subrules.

In (16), Iface is a finite-state relation that injects dependency brackets to the parses according to the

dependency rules.

$$\text{Iface} = \text{Intro} \circ \text{Chk}, \quad (16)$$

$$\text{Intro} = (\text{Id}(\Omega^*\Pi^*)(\epsilon\times\Gamma^*)\text{Id}(\#))^*, \quad (17)$$

$$\text{Chk} = \text{Proj}_1(\text{Match} \circ \text{Rules}), \quad (18)$$

$$\text{Rules} = \text{Id}\left(\cup_{i=1}^{r} F_i G_{i\#}\right)^* . \quad (19)$$

$$\text{Match} = (\text{Id}(\Omega^*)\ \text{Mid}\ \text{Id}(\Omega^*)\ \text{Tag}^*\ \text{Id}(\#))^* \quad (20)$$

$$\text{Mid} = \text{Id}(\epsilon) \cup (\Omega^* \times \%), \quad (21)$$

$$\text{Tag} = \text{Id}(\Pi) \cup (\Pi \times \epsilon). \quad (22)$$

Iface is the composition of relations Intro and Chk. Relation Intro inserts dependency brackets between the morphological analysis of each token and the following token boundary. Relation Chk verifies that the inserted brackets are supported by dependency rules that are represented by relation Rules.

In order to allow generalizations in the specification of morphological categories, the relation Intro does not match dependency rules directly, but through a filter. This filter, Match, optionally replaces the middle part of each lexeme with % and arbitrary morphological feature labels with the empty string.

In addition to the dependency rules, we need to define the semantics of the root rules. Let H be the set of the categories having a root rule. The category of the root word will be indicated in the dependency bracketing as an unmatched bracket. It is checked by relation $\text{Root} = \text{Id}(H\#)$ that replaces $\text{Root} = \text{Id}(\#)$ in the composition formulas (1) .

3.5 An Efficient Implementation

The definition of Iface gives rise to a naive parser implementation that is based on the formula

$$\text{Parses}(x) = \text{MI}_x \circ \text{Chk} \circ \text{Syn}_{n-1}, \quad (23)$$

$$\text{MI}_x = \text{Id}(x) \circ \text{Morph} \circ \text{Intro}. \quad (24)$$

The naive implementation is inefficient in practice. The main efficiency problem is that the state complexity of relation Chk can be exponential to the number of rules. To avoid this, we replace it with Chk_x, a restriction of Chk. This restriction is computed lazily when the input is known.

$$\text{Parses}(x) = \text{MI}_x \circ \text{Chk}_x \circ \text{Syn}_{n-1}, \quad (25)$$

$$\text{Chk}_x = \text{Proj}_1(\text{Match}_x \circ \text{Rules}) \quad (26)$$

$$\text{Match}_x = \text{Proj}_2(\text{MI}_x) \circ \text{Match}. \quad (27)$$

In this improved method, the application of Iface demands only linear space according to the number of rules. This method is also fast to apply to the input, as far as the morphology-syntax interface is concerned. Meanwhile, one efficient implementation of Syn_{n-1} is already provided in (Yli-Jyrä, 2012).

4 The Most Specific Parse

The parsing method of (Yli-Jyrä, 2012) builds the parse forest efficiently using several transducers, but there is no guarantee that the whole set of parses could be extracted efficiently from the compact representation constructed during the recognition phase. We will now assume, however, that the number of parses is, in practice, substantially smaller than in the theoretically possible worst case. Moreover, it is even more important to assume that the set of parses is compactly packed into a finite automaton. These two assumptions let us proceed by refining the parse forest without using weights such as in (Yli-Jyrä, 2012).

In the following, we restrict the parse forest to those parses that have the smallest number of 'linear successor' dependencies (++). The number of such dependencies is compared with a finite-state relation $\text{Cp} \subseteq (\Gamma \cup \{\#\})^* \times (\Gamma \cup \{\#\})^*$ constructed as follows:

$$\Sigma' = \Sigma - \{\text{++>}\}, \quad (28)$$

$$\text{Cp} = \text{Map}_i \circ (\text{Id}(\text{++>}^*)(\epsilon \times \text{++>})^+) \circ \text{Map}_i^{-1}, \quad (29)$$

$$\text{Map}_i = (\text{Id}(\text{++>}) \cup (\Sigma' \times \epsilon))^*. \quad (30)$$

In practice, the reduction of the parse forest is possible only if the parse forest $\text{Proj}_2(\text{Parses}(x))$ is recognized by a sufficiently small finite-state automaton that can then be operated in Formula (33). The parses that minimize the number of 'linear successor' dependencies are obtained as the output of the relation $\text{Parses}'(x)$.

$$\text{Parses}'(x) = \text{MI}_x \circ \text{Chk}_x \circ T_{x,1}, \quad (31)$$

$$\text{T}_{x,0} = \text{Proj}_2(\text{Parses}(x)), \quad (32)$$

$$\text{T}_{x,1} = \text{T}_{x,0} - \text{Proj}_2(\text{T}_{x,0} \circ \text{Cp} \circ \text{T}_{x,0}). \quad (33)$$

This restriction technique could be repeatedly applied to further levels of specificity. For example, lexically motivated complements could be preferred over adjuncts and other grammatically possible dependents.

5 Evaluation and Discussion

5.1 Elegance

We have retained most of the elegancy in the contracting finite-state dependency parser (Yli-Jyrä, 2012). The changes introduced in this paper are modular and implementable with standard operations on finite-state transducers.

Our refined design for a parser can be implemented largely in similar lines as the general approach (Yli-Jyrä, 2012) up to the point when the parses are extracted from the compact parse forest.

Parsing by arc contractions is closely related to the idea of reductions with restarting automata (Plátek et al., 2003).

5.2 Coverage

The representation of the parses can be extended to handle word-internal token boundaries, which facilitates the adequate treatment of agglutinative languages, cf. (Oflazer, 2003).

The limit for nested brackets is based on the psycholinguistic reality (Miller, 1956; Kornai and Tuza, 1992) and the observed tendency for short dependencies (Lin, 1995; Eisner and Smith, 2005) in natural language.

The same general design can be used to produce non-projective dependency analyses as required by many European languages. The crossing dependencies can be assigned to two or more planes as suggested in (Yli-Jyrä, 2012). 2-planar bracketing already achieves very high recall in practice (Gómez-Rodríguez and Nivre, 2010).

5.3 Ambiguity Management

Oflazer (2003) uses the lenient composition operation to compute the number of bottom-up fragments in incomplete parses. The current solution improves above this by supporting gapped fragments and unrestricted counting of the graph components.

Like in another extended finite-state approach (Oflazer, 2003), the ambiguity in the output of our parsing method can be reduced by removing parses with high total link length and by applying filters that enforce barrier constraints to the dependency links.

5.4 Computational Complexity

Thanks to dynamically applied finite-state operations and the representation of feature combinations as strings rather than regular languages, the dependency rules can be compiled quickly into the transducers used by the parser. For example, the actual specifications of dependency rules are now compiled into a linear-size finite-state transducer, Chk. The proposed implementation for the morphology-syntax interface is, thus, a significant improvement in comparison to the common approach that compiles and combines replacement rules into a single transducer where the morphological conditions of the rules are potentially mixed in a combinatorial manner.

Although we have started to write an experimental grammar, we do not exactly know how many rules a mature grammar will contain. Lexicalization of the rules will increase the number of rules significantly. The number of syntactic categories will increase even more if complements are lexicalized.

5.5 Robustness

In case the grammar does not fully disambiguate or build a complete dependency structure, the parser should be able to build and produce a partial analysis. (In interactive treebanking, it would be useful if an additional knowledge source, e.g. a human, can be used to provide additional information to help the parser carry on the analysis to a complete structure.)

The current grammar system indeed assumes that it can build complete trees for all input sentences. This assumption is typical for all generative grammars, but seems to contradict the requirement of robustness. To support robust parsing, we have now proposed a simple technique where partial analyses are connected into a tree with the "linear successor" links. The designed parser tries its best to avoid these underspecific links, but uses the smallest possible number of them to connect the partial analyses into a tree if more grammatical parses are not available.

5.6 Future Work

Although Oflazer (2003) does not report significant problems with long sentences, it may be difficult to construct a single automaton for the parse forest of a

sentence that contains many words. In the future, a more efficient method for finding the most specific parse from the forest can be worked out using weighted finite-state automata. Such a method would combine the approaches of the companion paper (Yli-Jyrä, 2012) and the current paper.

It seems interesting to study further how the specificity reasoning and statistically learned weights could complement each other in order to find the best analyses. Moreover, the parser can be modified in such a way that debugging information is produced. This could be very useful, especially when learning contractions that handle the crossing dependencies of non-projective trees.

A dependency parser should enable the building of multiple types of analyses, e.g. to account for syntactic and semantic dependencies. Also adding more structure to the syntactic categories could be useful.

6 Conclusions

The current theoretical work paves the way for a full parser implementation. The parser should be able to cope with large grammars to enable efficient development, testing and application cycles.

The current work has sketched an expressive and compact formalism and its efficient implementation for the morphology-syntax interface of the contracting dependency parser. In addition, the work has elaborated strategies that help to make the grammar more robust without sacrificing the optimal specificity of the analysis.

Acknowledgments

The research has received funding from the Academy of Finland under the grant agreement # 128536 and the FIN-CLARIN project, and from the European Commission's 7th Framework Program under the grant agreement # 238405 (CLARA).

References

Jason Eisner and Noah A. Smith. 2005. Parsing with soft and hard constraints on dependency length. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, pages 30–41, Vancouver, October.

David Elworthy. 2000. A finite state parser with dependency structure output. In *Proceedings of Sixth International Workshop on Parsing Technologies (IWPT 2000*, Trento, Italy, February 23-25. Institute for Scientific and Technological Research.

Haim Gaifman. 1965. Dependency systems and phrase-structure systems. *Information and Control*, 8:304–37.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492—-1501, Uppsala, Sweden, 11-16 July.

David G. Hays. 1964. Dependency theory: A formalism and some observations. *Language*, 40:511–525.

Fred Karlsson, Atro Voutilainen, Juha Heikkiä, and Arto Anttila, editors. 1995. *Constraint Grammar: a Language-Independent System for Parsing Unrestricted Text*, volume 4 of *Natural Language Processing*. Mouton de Gruyter, Berlin and New York.

András Kornai and Zsolt Tuza. 1992. Narrowness, pathwidth, and their application in natural language processing. *Discrete Applied Mathematics*, 36:87–92.

Dekang Lin. 1995. A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25, 1995*, volume 2, pages 1420–1425.

George A. Miller. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):343–355.

Kemal Oflazer. 2003. Dependency parsing with an extended finite-state approach. *Computational Linguistics*, 29(4):515–544.

Martin Plátek, Markéta Lopatková, and Karel Oliva. 2003. Restarting automata: motivations and applications. In M. Holzer, editor, *Workshop 'Petrinetze' and 13. Theorietag 'Formale Sprachen und Automaten'*, pages 90–96, Institut für Informatik, Technische Universität München.

Anssi Yli-Jyrä. 2005. Approximating dependency grammars through intersection of star-free regular languages. *International Journal of Foundations of Computer Science*, 16(3).

Anssi Yli-Jyrä. 2012. On dependency analysis via contractions and weighted FSTs. In Diana Santos, Krister Lindén, and Wanjiku Ng'ang'a, editors, *Shall we Play the Festschrift Game? Essays on the Occasion of Lauri Carlson's 60th Birthday*. Springer-Verlag, Berlin.